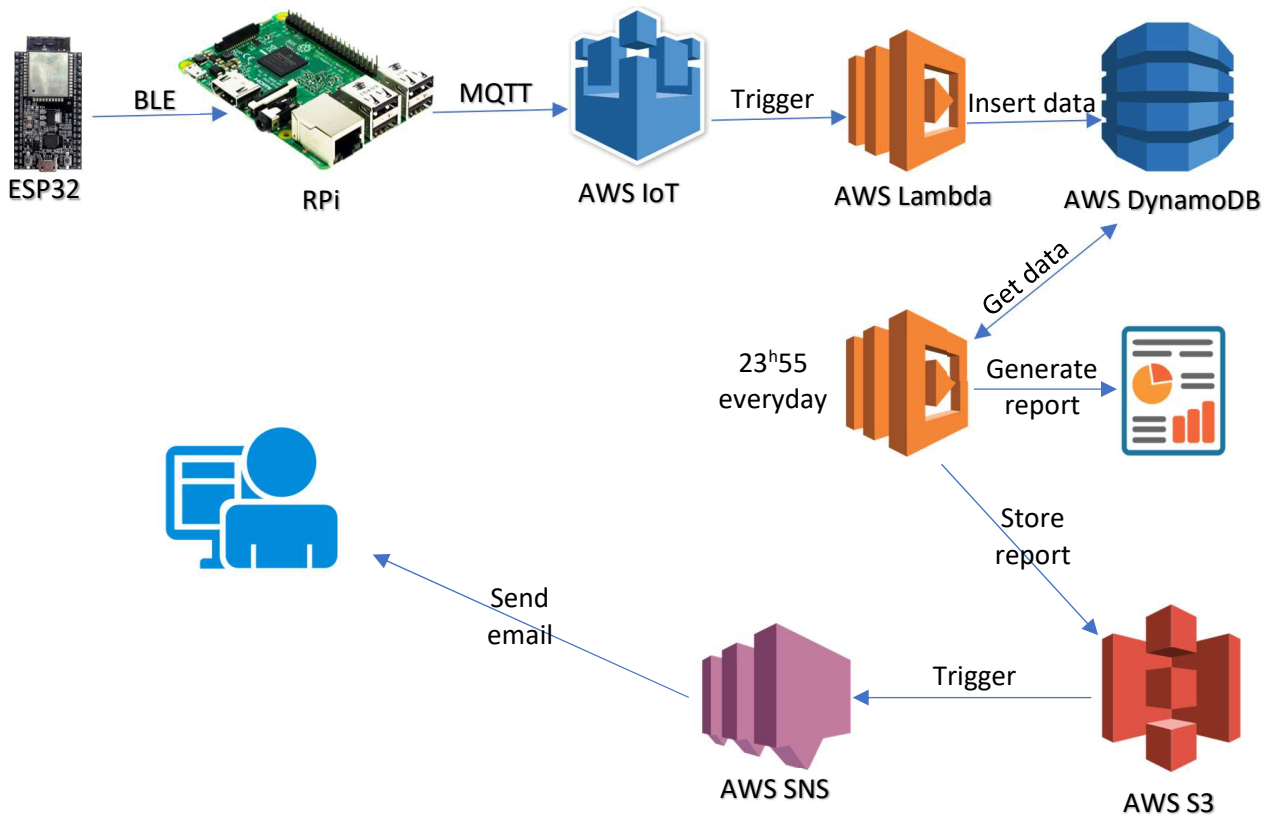
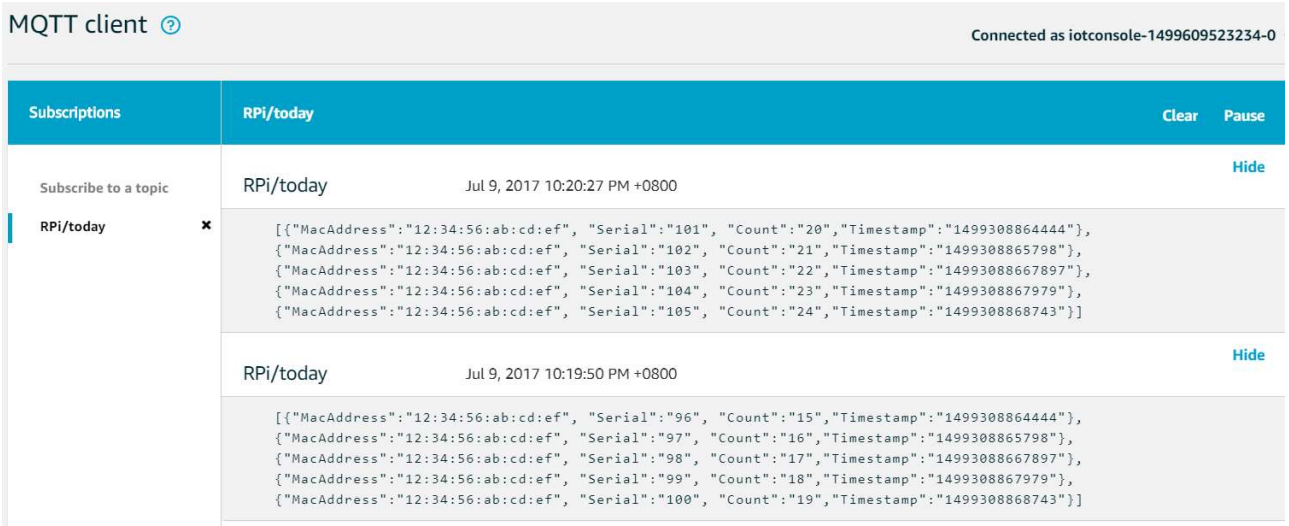
1. Server side
    a. General process



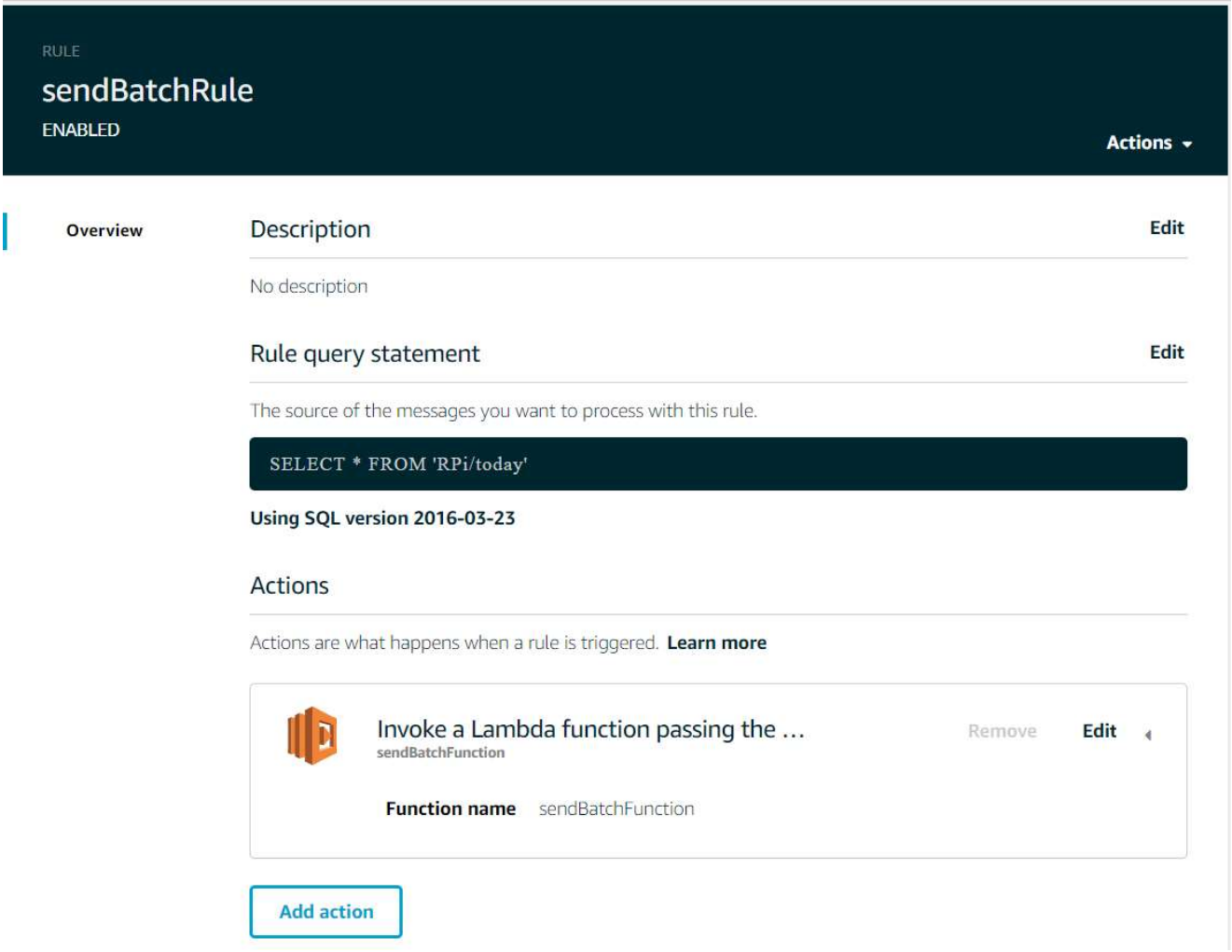
b. RPi → AWS IoT (send data through MQTT)

RPi(s) publish MQTT messages on topic (e.g. RPi/today), so we can subscribe to the topic to view the messages published to it. (more details in Hardware side (Dat))



c. AWS IoT → AWS DynamoDB

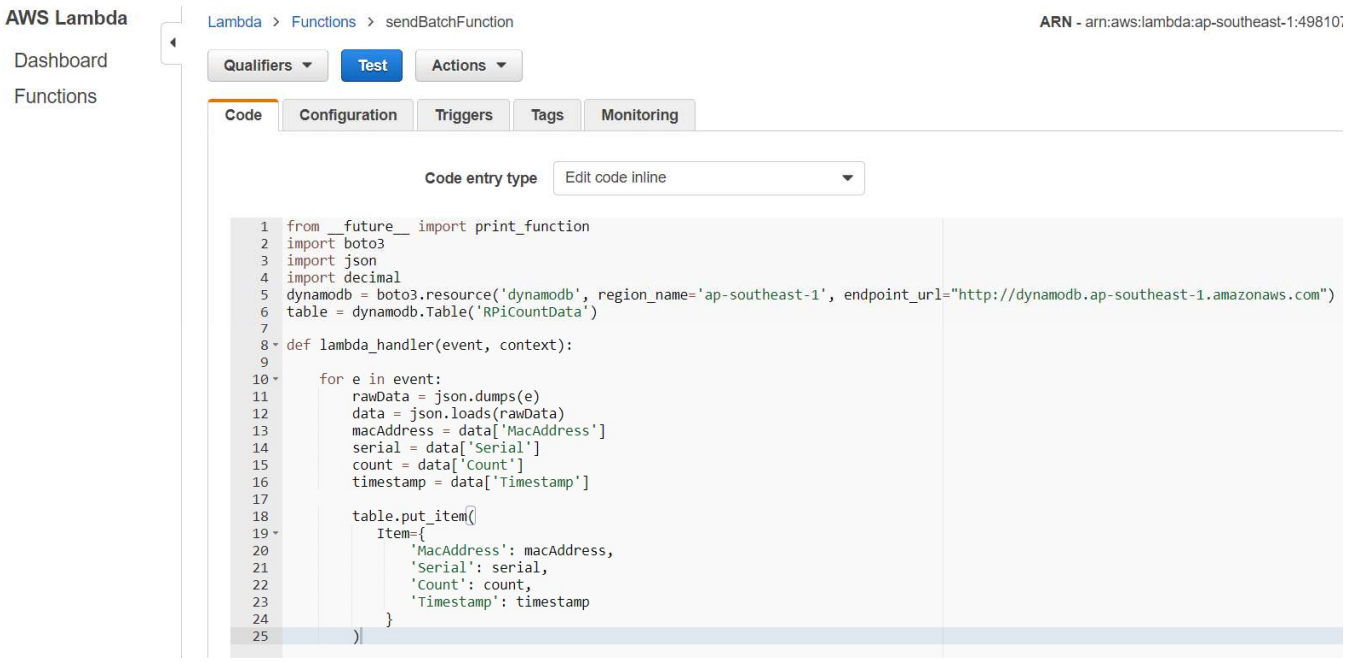
We create a rule for AWS IoT so whenever AWS IoT receive messages from RPi(s), a Lambda function will be executed.



The rule will take a message AWS IoT just receive to be a source for action, so we state a query:

" SELECT * from 'RPi/today' "

And add a Lambda function as an action.



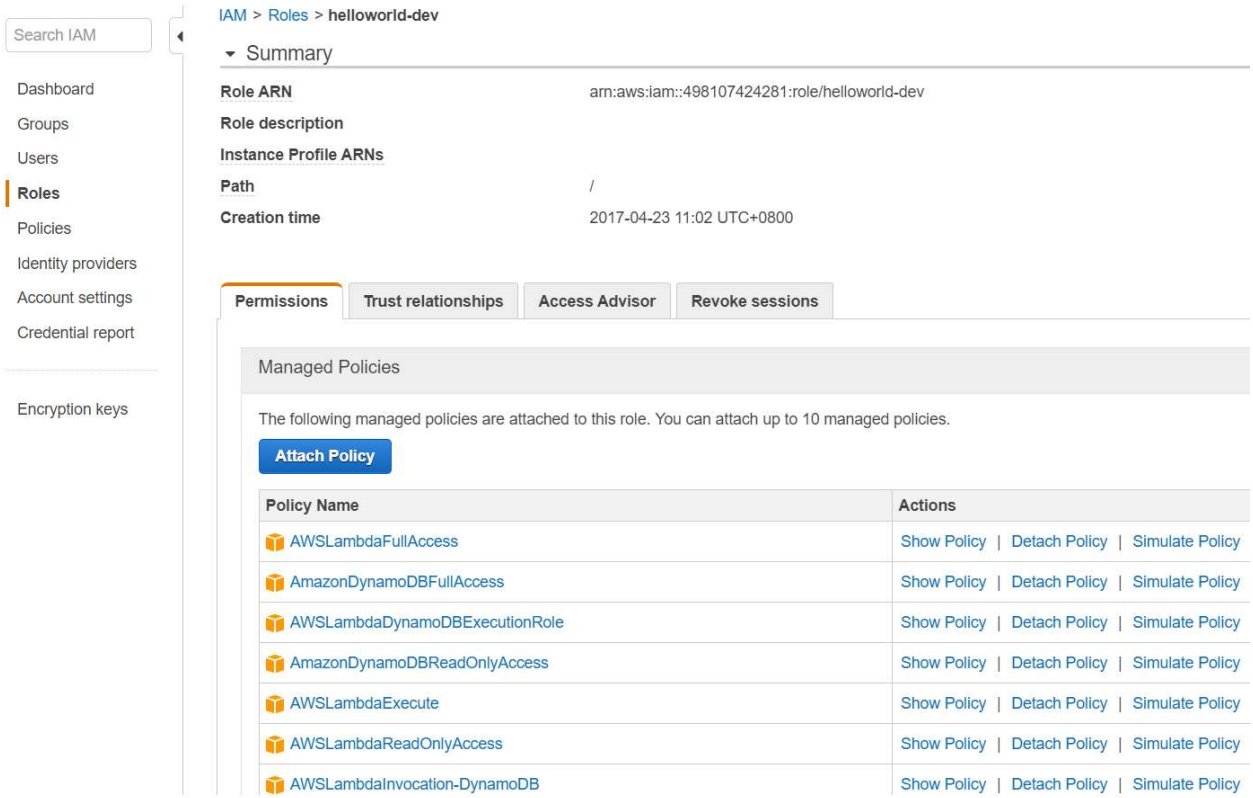
In the function, we connect to DynamoDB's table by:

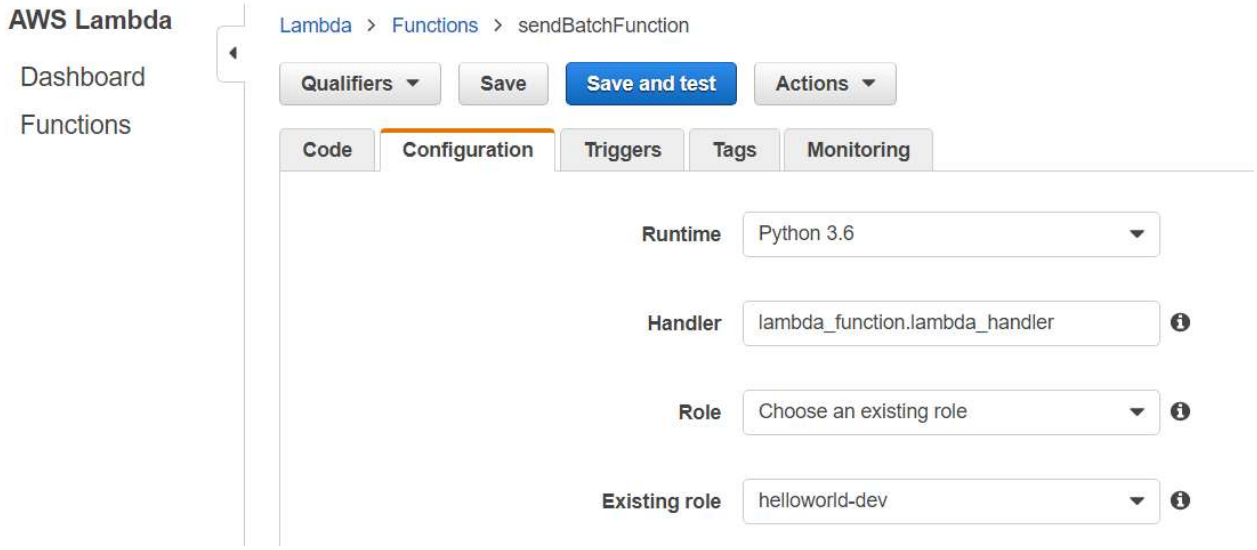
```
dynamodb = boto3.resource('dynamodb', region_name='ap-southeast-1', endpoint_url="http://dynamodb.ap-southeast-1.amazonaws.com")
table = dynamodb.Table('RPiCountData')
```

Data in the message that we query above will be in "event" variable, so we convert it into array of JSON data and insert data into DynamoDB table.

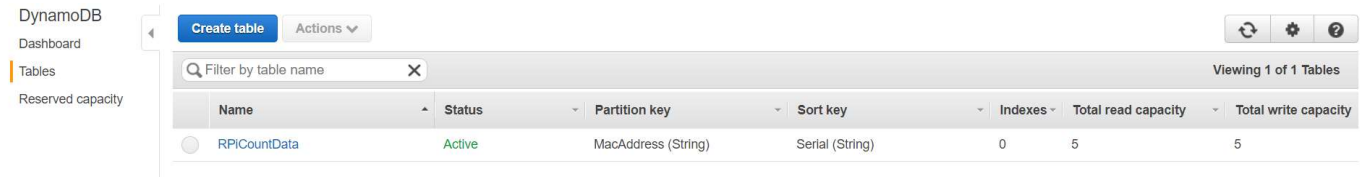
```
def lambda_handler(event, context):
    for e in event:
        rawData = json.dumps(e)
        data = json.loads(rawData)
        macAddress = data['MacAddress']
        serial = data['Serial']
        count = data['Count']
        timestamp = data['Timestamp']

        #Insert data into DynamoDB table
        table.put_item(
            Item={
                'MacAddress': macAddress,
                'Serial': serial,
                'Count': count,
                'Timestamp': timestamp
            }
        )
```

We also must create a role and give that role permission (by attach policies) to execute the function.

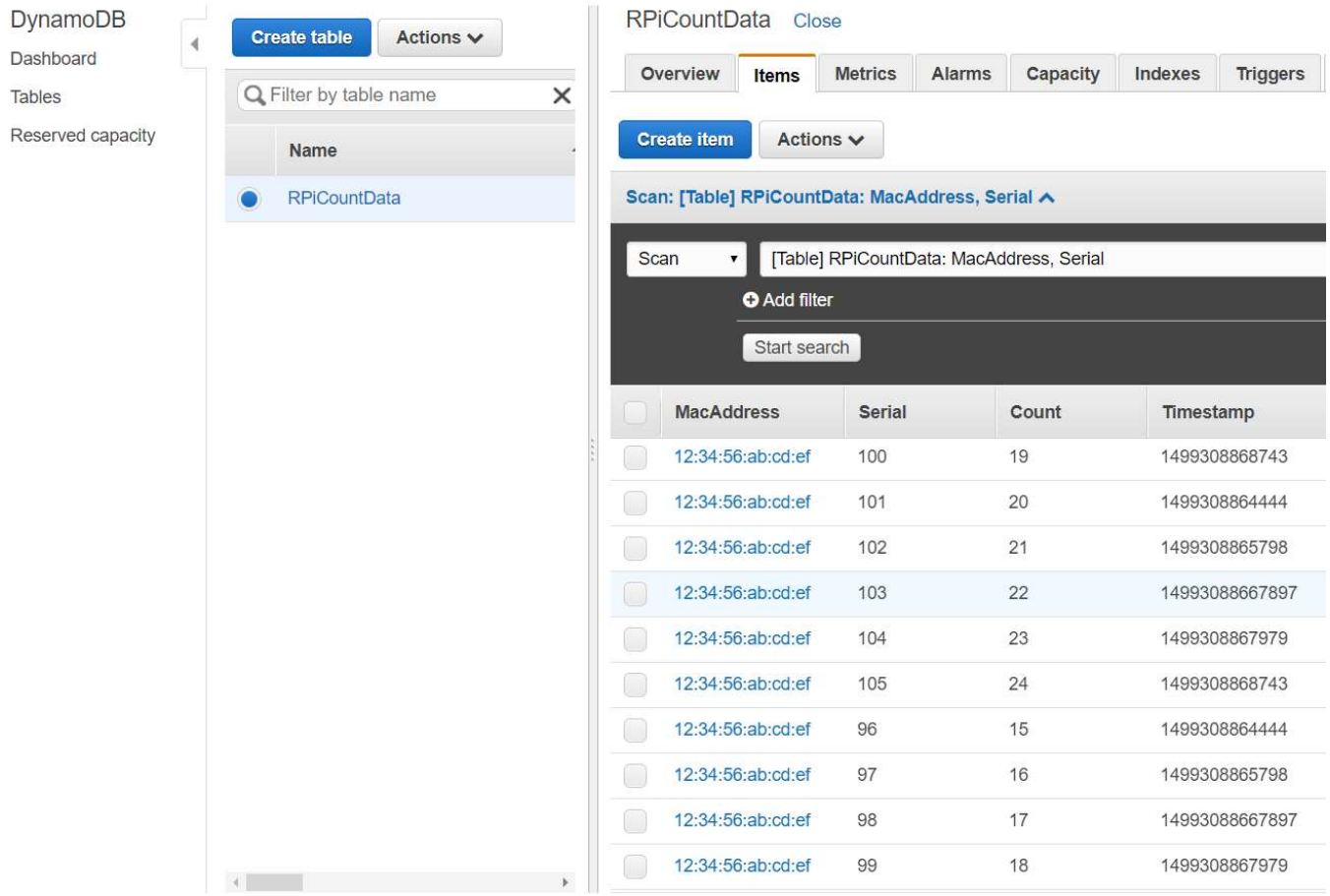




In DynamoDB, we create a table to store data.



And then whenever RPi(s) send message to AWS IoT, data will be insert into DynamoDB table:



So, all the process will be:

- Create a AWS role, attach policies.
- Create a DynamoDB table.
- Create a Lambda function to process incoming data from RPi and update them to DynamoDB table, configure function's role with role created above.
- Subscribe to MQTT client topic to view incoming data from RPi.
- Create a rule so whenever a MQTT topic have new coming message, it will excute the Lambda function above.

d. Create a Lambda function to get data from DynamoDB's table and generate report at the end of the day

Services Resource Groups

```python
from __future__ import print_function
import boto3
import json
import decimal
from boto3.dynamodb.conditions import Key, Attr
from datetime import datetime, date, time

# Helper class to convert a DynamoDB item to JSON.
class DecimalEncoder(json.JSONEncoder):
    def default(self, o):
        if isinstance(o, decimal.Decimal):
            if o % 1 > 0:
                return float(o)
            else:
                return int(o)
        return super(DecimalEncoder, self).default(o)



def lambda_handler(event, context):
    dynamodb = boto3.resource('dynamodb', region_name='ap-southeast-1', endpoint_url="http://dynamodb.ap-southeast-1.amazonaws.com")
    table = dynamodb.Table('RPiCountData')

    midnight = datetime.combine(date.today(), time.min)
    tist = str(datetime.strptime(str(midnight), "%Y-%d-%m %H:%M:%S").timestamp()*1000)
    fe = Key('Timestamp').gt(tist);

    pe = "MacAddress, Serial, #co, #ti"
    ean = {"#co": "Count", "#ti":"Timestamp"}
    esk = None

    response = table.scan(
        FilterExpression=fe,
        ProjectionExpression=pe,
        ExpressionAttributeNames=ean
    )

    for i in response['Items']:
        print(json.dumps(i, cls=DecimalEncoder))

    while 'LastEvaluatedKey' in response:
        response = table.scan(
            ProjectionExpression=pe,
            FilterExpression=fe,
            ExpressionAttributeNames= ean,
            ExclusiveStartKey=response['LastEvaluatedKey']
            )
        for i in response['Items']:
            print(json.dumps(i, cls=DecimalEncoder))
    return 'Successful'
```

Feedback English © 2008 - 2017, Amazon Web Ser

For easier process data get from DynamoDB's table, we convert it into JSON:

```python
# Helper class to convert a DynamoDB item to JSON.
class DecimalEncoder(json.JSONEncoder):
    def default(self, o):
        if isinstance(o, decimal.Decimal):
            if o % 1 > 0:
                return float(o)
            else:
                return int(o)
        return super(DecimalEncoder, self).default(o)
```

Connect to DynamoDB's table:

```
dynamodb = boto3.resource('dynamodb', region_name='ap-southeast-1', endpoint_url="http://dynamodb.ap-southeast-1.amazonaws.com")
table = dynamodb.Table('RPiCountData')
```

To get all data of the last day, we create a midnight timestamp of the day before and compare it with timestamp field of all data, and just get data satisfied (greater than last midnight timestamp):

```
    midnight = datetime.combine(date.today(), time.min)
    tist = str(datetime.strptime(str(midnight), "%Y-%d-%m %H:%M:%S").timestamp()*1000)
    fe = Key('Timestamp').gt(tist);

    pe = "MacAddress, Serial, #co, #ti"
    ean = {"#co": "Count", "#ti":"Timestamp"}
    esk = None

    response = table.scan(
        FilterExpression=fe,
        ProjectionExpression=pe,
        ExpressionAttributeNames=ean
    )
```

ProjectionExpression (pe) specifies the attributes we want in the scan result.

FilterExpression (fe) specifies a condition that returns only items that satisfy the condition. All other items are discarded.

ExpressionAttributeNames (ean) provides name substitution, we use this because "count" and "timestamp" is a reserved word in DynamoDB, we cannot use it directly in any expression, so we use the expression attribute name "#co" and "#ti" to address this.

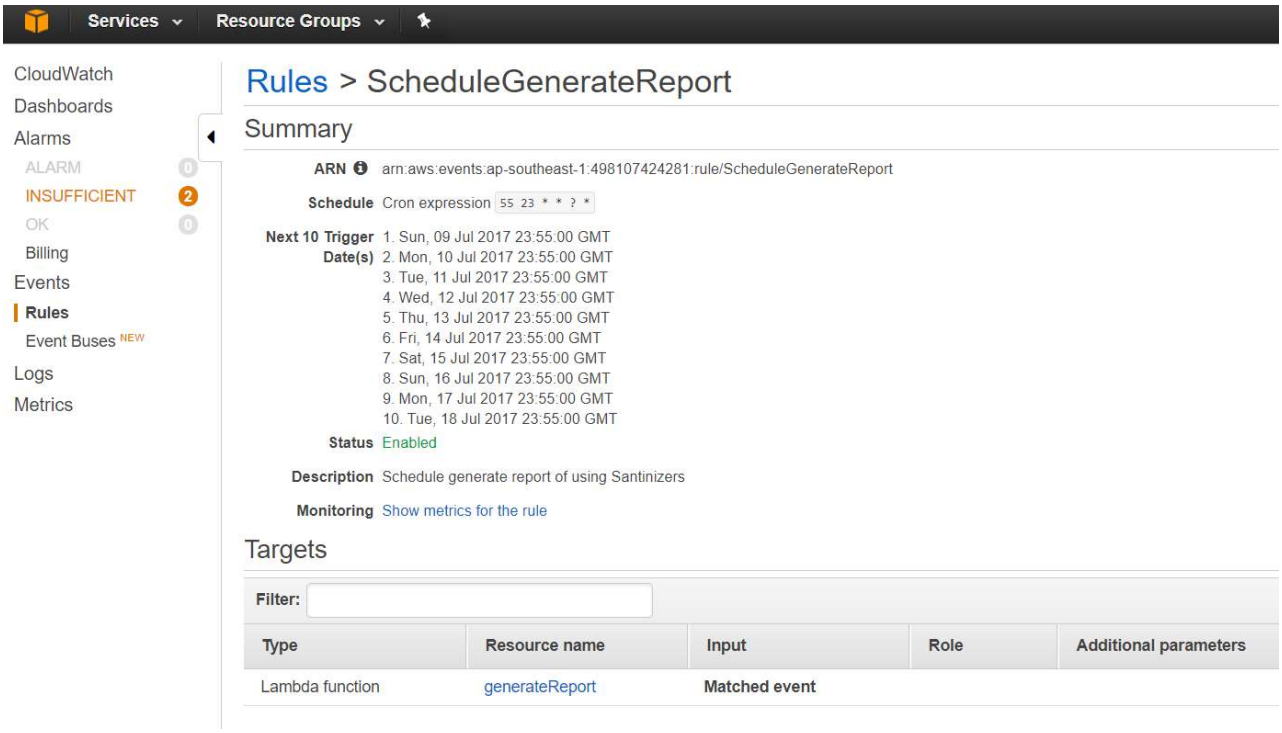
The scan method returns a subset of the the items each time, called a page. The LastEvaluatedKey value in the response is then passed to the scan method via the ExclusiveStartKey parameter. When the last page is returned, LastEvaluatedKey is not part of the response:

```
    while 'LastEvaluatedKey' in response:
        response = table.scan(
            ProjectionExpression=pe,
            FilterExpression=fe,
            ExpressionAttributeNames= ean,
            ExclusiveStartKey=response['LastEvaluatedKey']
            )
        for i in response['Items']:
            print(json.dumps(i, cls=DecimalEncoder))
```

All data we get now in "respone['Items']" as JSON and we can generate report from it.

We have not implemented code to generate report yet due to format problem.

To schedule the Lambda function executed at 23^{h}55 daily, we use AWS CloudWatch to do it



About the report, we can think about some questions like:

- Which time of day the sanitizers are used most?
- Which venue the sanitizers are used most?
- Which day the sanitizers are used most?
- List of all sanitizers which is less than 30% of volume.

…